

Class06: R Functions

Nathalie Huang (PID: A19134713)

Table of contents

Background	1
A first function	1
A second function	3
A new cool function	6

Background

Functions are at the heart of using R. Everything we do involves calling and using functions (from data input, analysis to results output).

All functions in R have at least 3 things:

1. A **name** the thing we use to call the function.
2. One or more input **arguments** that are comma separated
3. The **body**, lines of code between curly brackets { } that does the work of the function.

A first function

Let's write a silly wee function to add some numbers:

```
add <- function(x) {  
  x + 1  
}
```

Let's try it out

```
add(100)
```

```
[1] 101
```

Will this work

```
add( c(100, 200, 300) )
```

```
[1] 101 201 301
```

Modify to be more useful and add more than just 1

```
add <- function(x, y=1) {  
  x + y  
}
```

```
add(100,10)
```

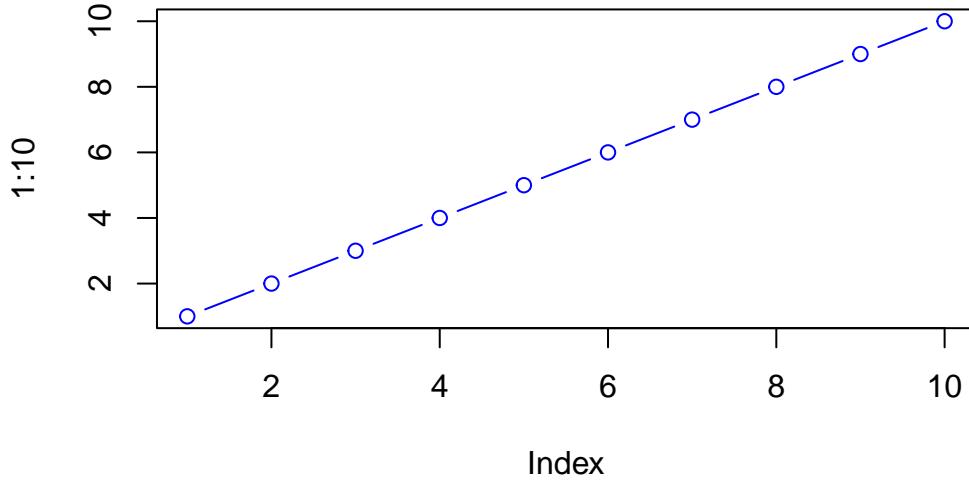
```
[1] 110
```

Will this work?

```
add(100)
```

```
[1] 101
```

```
plot(1:10, col="blue", typ="b")
```



```
log(10, base=10)
```

```
[1] 1
```

N.B. Input arguments can be either **required** or **optional**. The latter have a fall-back default that is specified in the function code with an equals sign.

```
#add(x=100, y=200, z=300)
```

A second function

All functions in R look like this

```
name <- function(arg) {
  body
}
```

The `sample()` function in R is used to generate random samples from a vector, which is useful for resampling, permutation tests, and simulations.

```
sample(1:10, size =4)
```

```
[1] 2 10 7 4
```

Q. Return 12 numbers picked randomly from the input 1:10

```
sample(1:10, size =12, replace = TRUE)
```

```
[1] 2 3 4 6 3 4 7 2 3 6 5 7
```

Q. Write the code to generate a 12 nucleotide long DNA sequence?

```
bases <- sample(  
  c("A", "C", "G", "T"), 12, replace = TRUE)
```

Q. Write a first version function called `generate_dna()` that generates a user specified length `n` random DNA sequence?

```
name <- function(arg) {  
  body  
}
```

```
generate_dna <- function(n=6) {  
  bases <- c("A", "C", "G", "T")  
  sample(bases, size=n, replace=TRUE)  
}
```

```
generate_dna(100)
```

```
[1] "T" "A" "A" "T" "T" "C" "T" "A" "G" "A" "C" "C" "C" "C" "T" "T" "C"  
[19] "T" "T" "G" "A" "G" "G" "G" "C" "G" "T" "T" "T" "G" "T" "T" "G" "C" "C"  
[37] "T" "G" "C" "A" "T" "C" "T" "T" "C" "C" "T" "A" "G" "A" "T" "C" "T" "C"  
[55] "G" "A" "C" "G" "T" "T" "G" "T" "G" "G" "A" "A" "C" "G" "G" "A" "G"  
[73] "A" "T" "C" "C" "C" "A" "T" "A" "T" "T" "T" "C" "A" "A" "C" "A" "T" "T"  
[91] "T" "T" "A" "T" "T" "G" "A" "A" "G" "T"
```

Q. Modify your function to return a FASTA like sequence so rather than [1] “G” “C” “A” “A” “T” we want “GCAAT”

```
generate_dna <- function(n=6) {
  bases <- c("A", "C", "G", "T")
  ans <- sample(bases, size=n, replace=TRUE)
  paste(ans, collapse = "")
  return(ans)
}
```

```
generate_dna(10)
```

```
[1] "A" "A" "T" "A" "C" "T" "A" "T" "C" "T"
```

An example

```
# Example pattern (not using your bases)
x <- c("H", "E", "L", "L", "O")

paste(x, collapse = "****")
```

```
[1] "H****E****L****L****O"
```

```
# returns "HELLO
```

Q. Give the user an option to return FASTA format output sequence or standard multi-element vector format?

```
generate_dna <- function(n=6, fasta=TRUE) {
  bases <- c("A", "C", "G", "T")
  ans <- sample(bases, size=n, replace=TRUE)

  if(fasta){
    ans <- paste(ans, collapse = "")
    cat("Hello...")
  } else{
    cat("...is it me you are looking for")
  }

  return(ans)
}
```

```
generate_dna(10)
```

Hello...

```
[1] "ATCTGATGTG"
```

```
generate_dna(10, fasta=F)
```

...is it me you are looking for

```
[1] "T" "T" "C" "T" "A" "C" "T" "C" "G" "C"
```

A new cool function

Q. Write a function called `generate_protein()` that generates a user specified length protein sequence in FASTA like format?

```
generate_protein <- function(n) {  
  aa <- c(  
    "A", "C", "D", "E", "F", "G", "H", "I", "K", "L",  
    "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y"  
  )  
  ans <- sample(aa, size = n, replace = TRUE)  
  return(paste(ans, collapse=""))  
}
```

```
generate_protein(10)
```

```
[1] "RVVGEDNWDM"
```

Q. Use your new `generate_protein()` function to generate all sequences between length 6 and 12 amino acids in length and check of any of these are unique in nature (i.e. found in the NR database at NCBI)?

```
generate_protein(6)
```

```
[1] "MGTDVS"
```

```
generate_protein(7)
```

```
[1] "NDRTPQG"
```

```
generate_protein(8)
```

```
[1] "DTFKIYFF"
```

```
generate_protein(9)
```

```
[1] "LNPHATLSF"
```

```
generate_protein(10)
```

```
[1] "QANATQQCIA"
```

```
generate_protein(11)
```

```
[1] "AEEKDQDWVNG"
```

```
generate_protein(12)
```

```
[1] "TMMNIQPTGQTN"
```

Or we could do a `for()` loop:

```
for(i in 6:12) {  
  cat(">", i, sep="", "\n")  
  cat(generate_protein(i), "\n")  
}
```

```
>6  
ATKYYI  
>7  
RFYFVGP  
>8
```

ESIGRVFE
>9
GWKYRPTQN
>10
DQPFLPMFVN
>11
MWPTKWSGQWF
>12
FTITSIFPSGHL